Huffman Coding

Static Huffman Coding

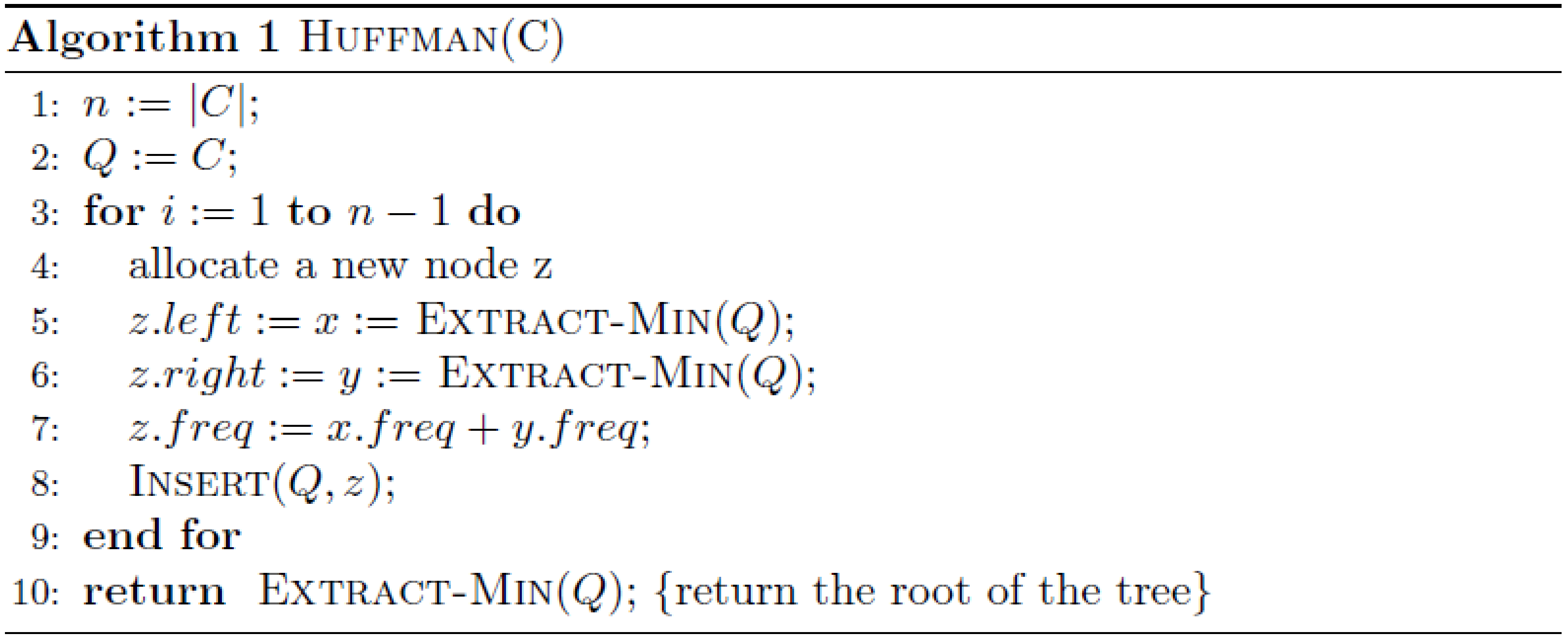
- Static Huffman coding assigns variable length codes to symbols based on their frequency of occurrences in the given message. Low frequency symbols are encoded using many bits, and high frequency symbols are encoded using fewer bits.

- The message to be transmitted is first analyzed to find the relative frequencies of its constituent characters.

- The coding process generates a binary tree, the Huffman code tree, with branches labeled with bits (0 and 1).

- The Huffman tree (or the character codeword pairs) must be sent with the compressed information to enable the receiver decode the message.

Static Huffman Coding Algorithm



Algorithm 1 HUFFMAN(C)

```
$n := |C|$;
$Q := C$;
for $i := 1$ to $n - 1$ do
    allocate a new node z
    $z.left := x :=$ EXTRACT-MIN($Q$);
    $z.right := y :=$ EXTRACT-MIN($Q$);
    $z.freq := x.freq + y.freq$;
    INSERT($Q, z$);
end for
return EXTRACT-MIN($Q$); {return the root of the tree}
```

Static Huffman Coding example

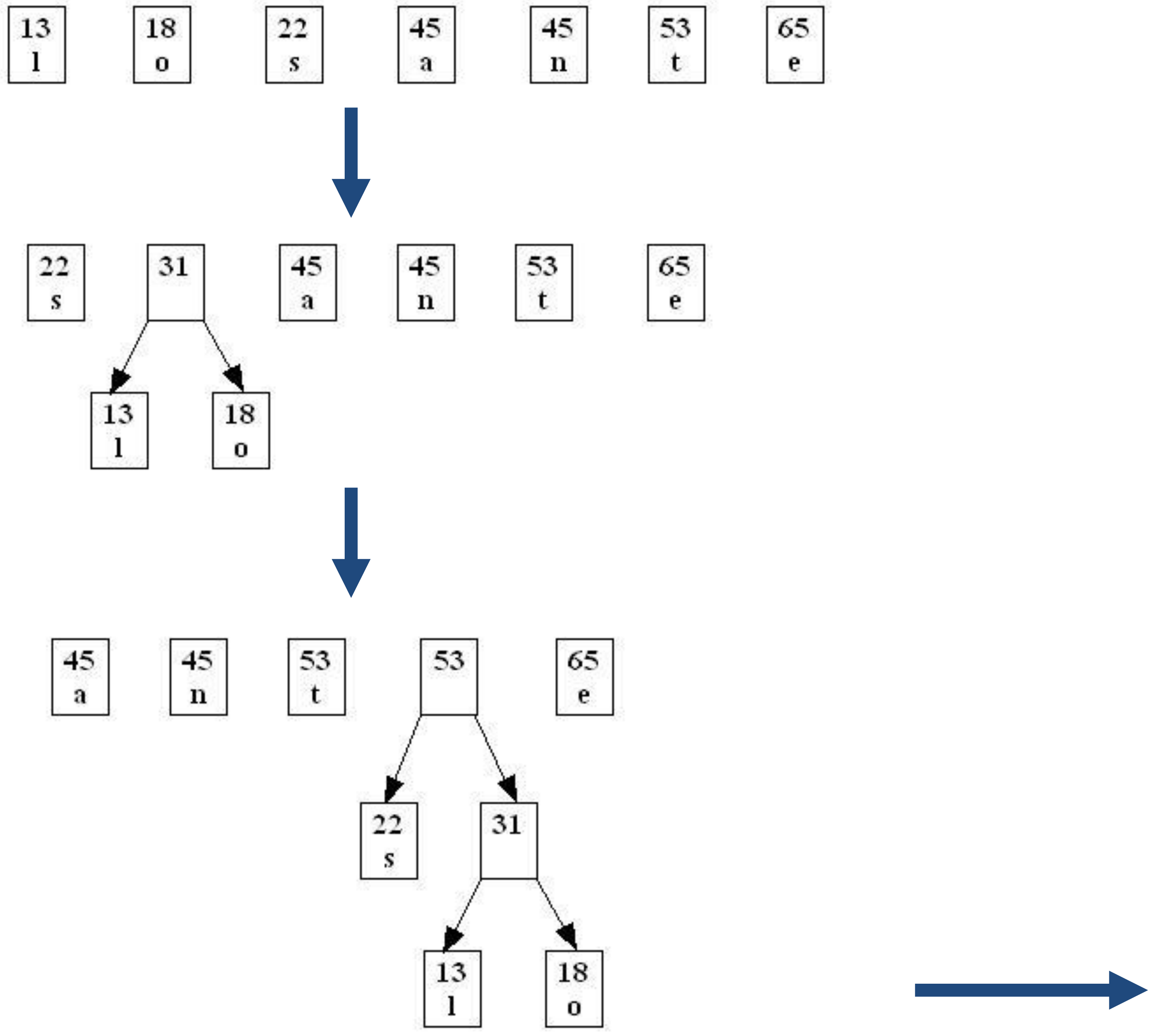
Example: Information to be transmitted over the internet contains the following characters with their associated frequencies:

Character	a	e	l	n	o	s	t
Frequency	45	65	13	45	18	22	53

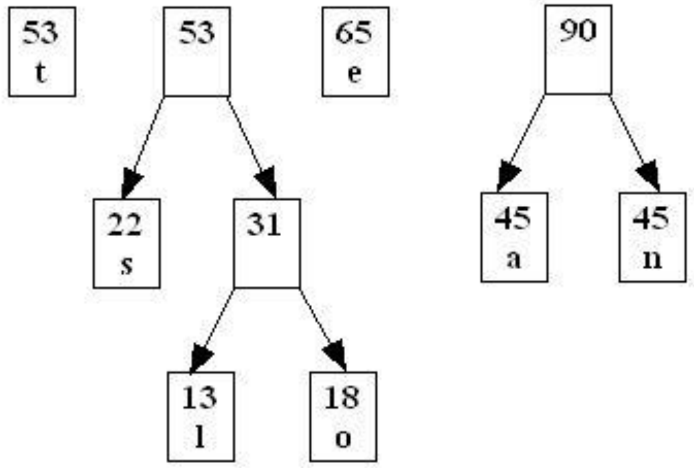
Use Huffman technique to answer the following questions:

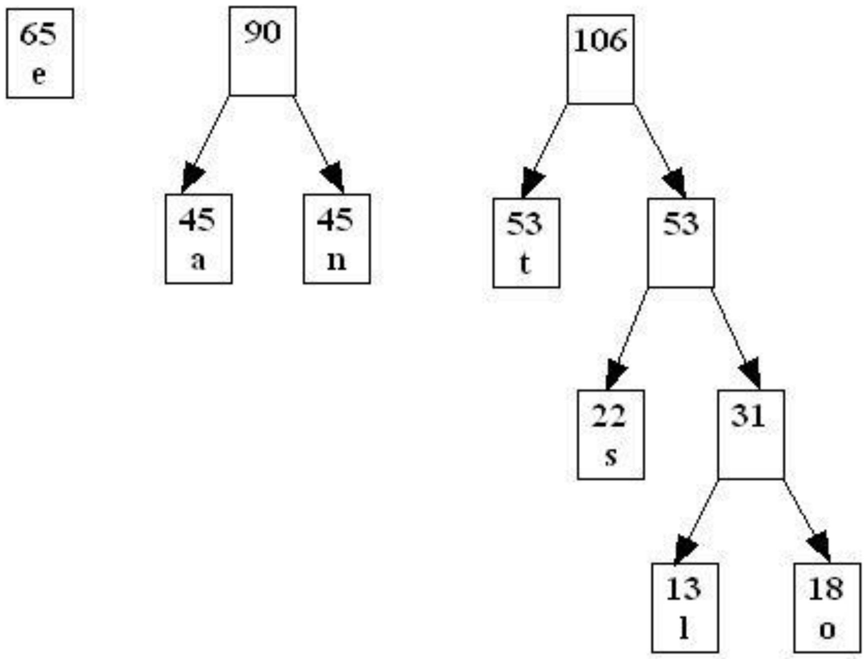
- ➢ Build the Huffman code tree for the message.
- ➢ Use the Huffman tree to find the codeword for each character.
- ➢ If the data consists of only these characters, what is the total number of bits to be transmitted? What is the compression ratio?
- ➢ Verify that your computed Huffman codewords satisfy the Prefix property.

Static Huffman Coding example (cont'd)

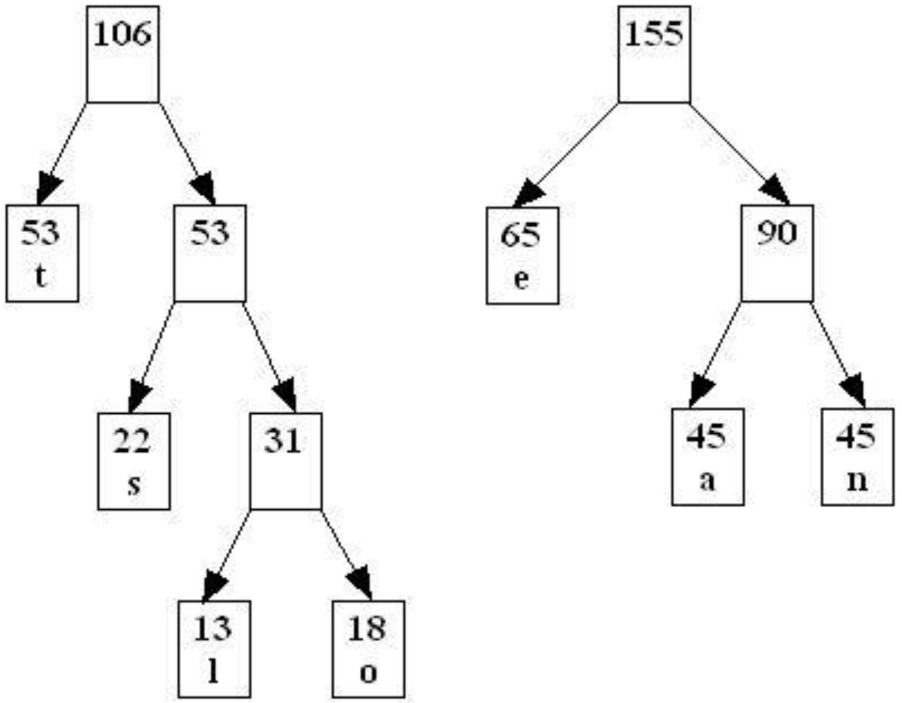


Static Huffman Coding example (cont'd)

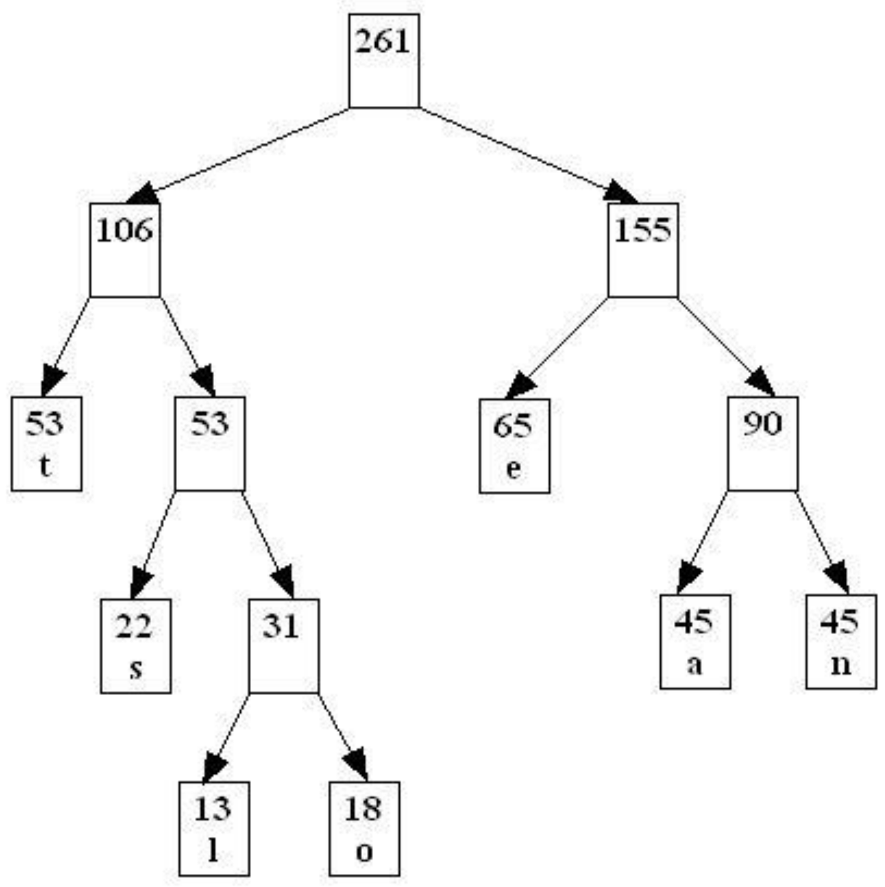




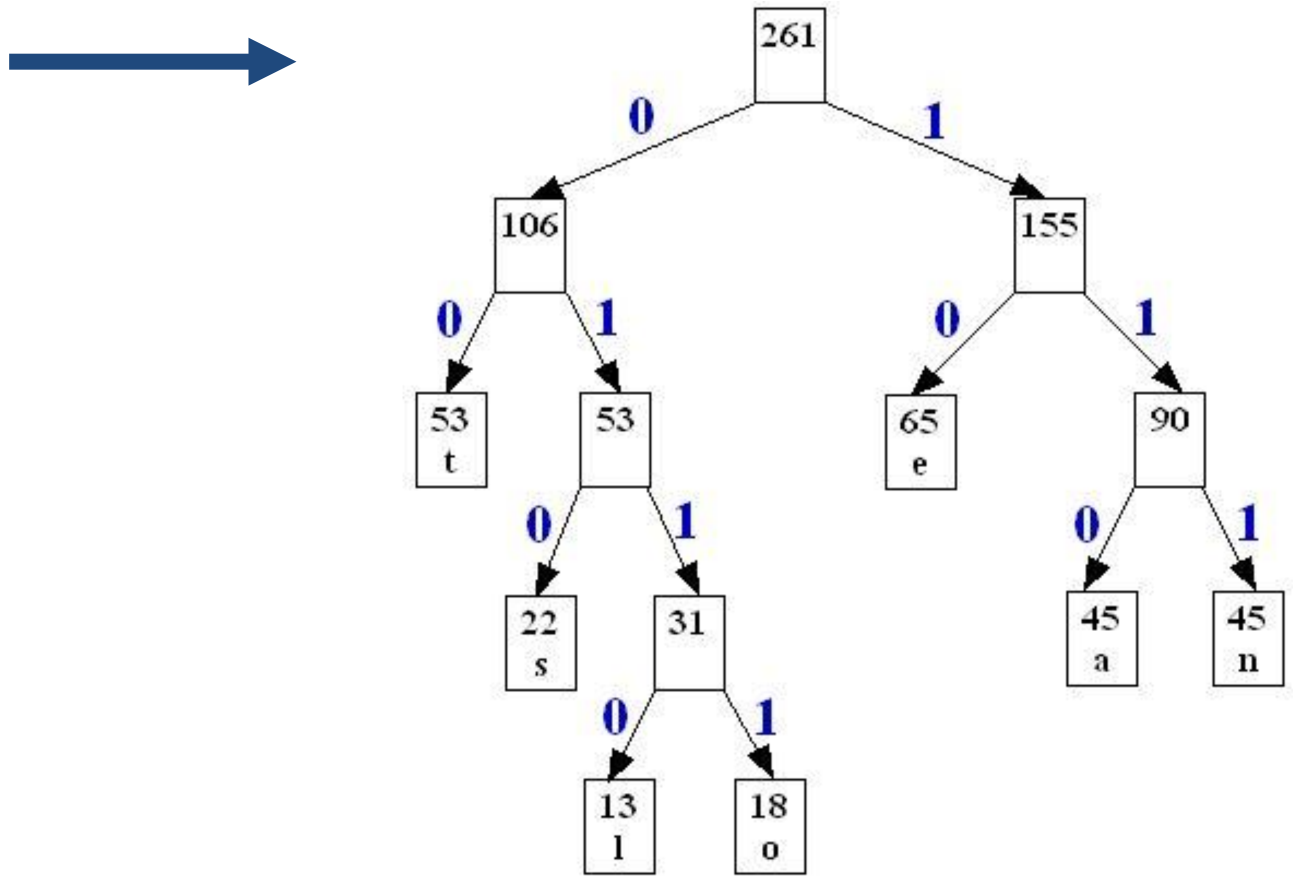
Static Huffman Coding example (cont’d)



Static Huffman Coding example (cont'd)



Static Huffman Coding example (cont'd)

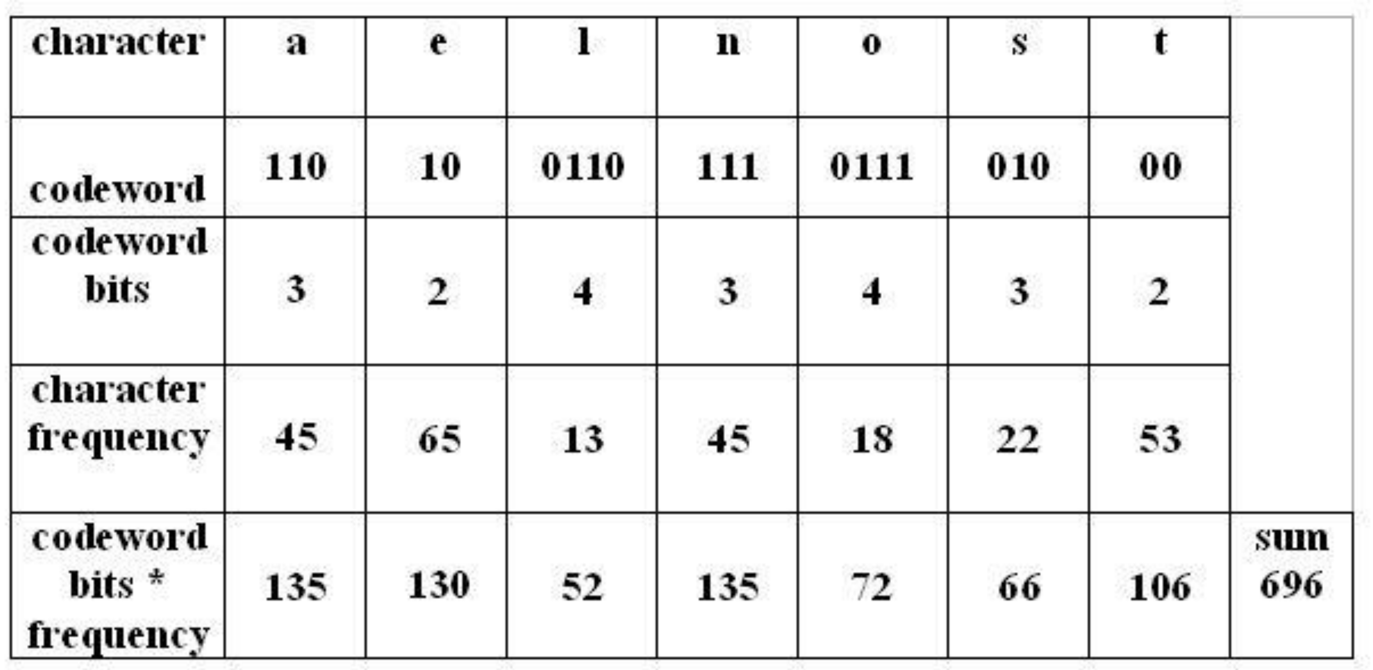


The sequence of zeros and ones that are the arcs in the path from the root to each leaf node are the desired codes:

character	a	e	l	n	o	s	t
Huffman codeword	**110**	**10**	**0110**	**111**	**0111**	**010**	**00**

Static Huffman Coding example (cont’d)

If we assume the message consists of only the characters a,e,l,n,o,s,t then the number of bits for the compressed message will be 696:



character	a	e	l	n	o	s	t	
codeword	110	10	0110	111	0111	010	00	
codeword bits	3	2	4	3	4	3	2	
character frequency	45	65	13	45	18	22	53	
codeword bits * frequency	135	130	52	135	72	66	106	sum 696

If the message is sent uncompressed with 8-bit ASCII representation for the characters, we have 261*8 = 2088 bits.

•

The Prefix Property

- Data encoded using Huffman coding is uniquely decodable. This is because Huffman codes satisfy an important property called the prefix property:

  In a given set of Huffman codewords, no codeword is a prefix of another Huffman codeword

- For example, in a given set of Huffman codewords, 10 and 101 cannot simultaneously be valid Huffman codewords because the first is a prefix of the second.

- We can see by inspection that the codewords we generated in the previous example are valid Huffman codewords.

The Prefix Property (cont’d)

To see why the prefix property is essential, consider the codewords given below in which “e” is encoded with **110** which is a prefix of “f”

character	a	b	c	d	e	f
codeword	0	101	100	111	110	1100

The decoding of 11000100110 is ambiguous:

11000100110 => **face**

11000100110 => **eaace**

Encoding and decoding examples

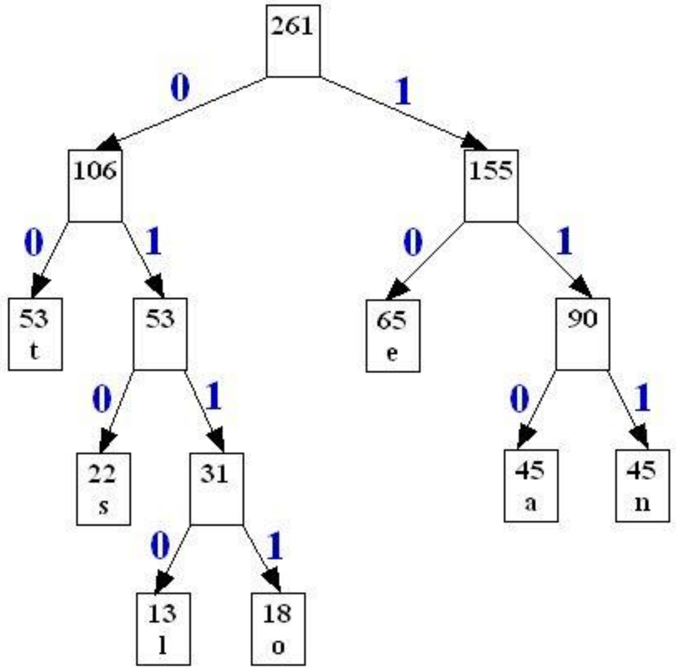
- Encode (compress) the message **tenseas** using the following codewords:

character	**a**	**e**	**l**	**n**	**o**	**s**	**t**
Huffman codeword	**110**	**10**	**0110**	**111**	**0111**	**010**	**00**

Answer: Replace each character with its codeword:

001011101010110010

- Decode (decompress) each of the following encoded messages, if possible, using the Huffman codeword tree given below **0110011101000** and **11101110101011**:



Answer: Decode a bit-stream by starting at the root and proceeding down the tree according to the bits in the message (0 = left, 1 = right). When a leaf is encountered, output the character at that leaf and restart at the root .If a leaf cannot be reached, the bit-stream cannot be decoded.

(a)0110011101000 => lost

(b) 11101110101011

The decoding fails because the corresponding node for 11 is not a leaf

Exercises

1. Using the Huffman tree constructed in this session, decode the following sequence of bits, if possible. Otherwise, where does the decoding fail?

   10100010111010001000010011

2. Using the Huffman tree constructed in this session, write the bit sequences that encode the messages:

   test , state , telnet , notes

   •

3. Mention one disadvantage of a lossless compression scheme and one disadvantage of a lossy compression scheme.

4. Write a Java program that implements the Huffman coding algorithm.